SSL needs to be get from third parties. It usually be a 2048 bit certificate that are widely distributed. The root certificate must be on end users machine in order to trust the certificate. If it not trusted , the browser will send an untrusted error message.

**Difference between SSL Certificate and Digital Certificate ?**

SSL is an Secure Socket Layer and it provides an standard way to establish the secure connection between client and server ( the data being transferred from and server).

Digital certificate is an electronic password that allows the client and server exchange the data securely using public key infrastructure (PKI).it is also known as public key digital certificate or digital certificate.

SSL is also called an Digital Certificate.

Secure connection , data encryption can be achieved by SSL. There are various types of encryption algorithms available.

* AES – Advanced Encryption Standard .it is an symmetric key encryption . need to share key to others for decryption . so less secure
* 3DES – Triple Data Encryption Standard. it encrypts data three times , so slow in process.
* RSA – one of the more secure cryptosystems widely used. SSL Use this mechnanism

It uses two key , public and private . Encrypt using public key and decrypt using the private key.

Payment Gateway – is an gateway which securely transact the payment information from ecommerce site to merchant’s site

**Spring Boot:**

* Which speeds up the application. It reduce lots of coding . No XML’s involved
* It can run as standalone or as an WAR.
* It can also be run by **mvn spring-boot:run**
* It requires java 1.7 or higher
* Current spring project inherit the **<spring-boot-dependency-parent>** project. It provides useful maven defaults. Add a ‘run’ goal that we use it in command line to start the application.
* Spring MVC dependency would be **<spring-boot-starter-web>**
* Jar can be created using the plugin **<spring-boot-maven-plugin>**
* Spring boot provides **SpringApplication** class to bootstrap the application

project xmlns="<a class="vglnk" href="<http://maven.apache.org/POM/4.0.0>" rel="nofollow"><span>http</span><span>://</span><span>maven</span><span>.</span><span>apache</span><span>.</span><span>org</span><span>/</span><span>POM</span><span>/</span><span>4</span><span>.</span><span>0</span><span>.</span><span>0</span></a>" xmlns:xsi="<a class="vglnk" href="<http://www.w3.org/2001/XMLSchema-instance>" rel="nofollow"><span>http</span><span>://</span><span>www</span><span>.</span><span>w3</span><span>.</span><span>org</span><span>/</span><span>2001</span><span>/</span><span>XMLSchema</span><span>-</span><span>instance</span></a>"

    xsi:schemaLocation="<a class="vglnk" href="<http://maven.apache.org/POM/4.0.0>" rel="nofollow"><span>http</span><span>://</span><span>maven</span><span>.</span><span>apache</span><span>.</span><span>org</span><span>/</span><span>POM</span><span>/</span><span>4</span><span>.</span><span>0</span><span>.</span><span>0</span></a> <a class="vglnk" href="<http://maven.apache.org/xsd/maven-4.0.0.xsd>" rel="nofollow"><span>http</span><span>://</span><span>maven</span><span>.</span><span>apache</span><span>.</span><span>org</span><span>/</span><span>xsd</span><span>/</span><span>maven</span><span>-</span><span>4</span><span>.</span><span>0</span><span>.</span><span>0</span><span>.</span><span>xsd</span></a>">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.websystique.springboot</groupId>

    <artifactId>SpringBootStandAloneExample</artifactId>

    <version>1.0.0</version>

    <packaging>**jar**</packaging>

    <name>SpringBootStandAloneExample</name>

**<parent>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-starter-parent</artifactId>**

**<version>1.4.3.RELEASE</version>**

**</parent>**

    <properties>

        <java.version>1.8</java.version>

    </properties>

    <dependencies>

**<!-- Add typical dependencies for a web application -->**

**<!-- Adds Tomcat and Spring MVC, along others -->**

**<dependency>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-starter-web</artifactId>**

**</dependency>**

**<dependency>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-starter-freemarker</artifactId>**

**</dependency>**

    </dependencies>

**<build>**

**<plugins>**

**<plugin><!-- Include if you want to make an executable jar[FAT JAR which**

**includes all dependencies along with sprinboot loader] that you can run on**

**commandline using java -jar NAME -->**

**<!-- If you are not including it, result would be just a regular jar.**

**You could run the project using mvn spring-boot:run [on project root directory]-->**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-maven-plugin</artifactId>**

**</plugin>**

**</plugins>**

**</build>**

</project>

* **@SpringBootApplication**  replace the **@Configuration , @EnableAutoConfiguration , @ComponentScan**
* Syntax is:

//OR JUST USE

**@SpringBootApplication(scanBasePackages={"com.websystique.springboot"})**

**/**/ same as @Configuration @EnableAutoConfiguration @ComponentScan combined

public class SpringBootStandAloneApp {

    public static void main(String[] args) {

**SpringApplication.run(**SpringBootStandAloneApp.class, args);

    }

}

* Springboot provides support for creating the WAR using the **SpringBootServletInitilizer.**
* **SpringBootServletInitilizer** is an abstract class which implements **WebApplicationInitilizer**
* Which configures the Servlet , Filter and ServletContextInitilizer bean from the application context to the servlet container.
* We should override the configure() method

public class SpringBootStandAloneWarApp extends SpringBootServletInitializer{

         @Override

    protected SpringApplicationBuilder **configure**(SpringApplicationBuilder application) {

        return application.sources(SpringBootStandAloneWarApp .class);

    }

    public static void main(String[] args) {

        SpringApplication.run(SpringBootStandAloneWarApp.class, args);

    }

}

**Spring Boot + REST API : (XML Rest Service)**

* Spring boot provides REST API support by providing the default dependencies and converters.
* Spring Boot REST API is same as Spring REST API.
* Any spring bean annotated with @RestController in spring boot application renders response in JSON as long as the Jackson2[Jackson-databind] in classpath.
* In spring boot we no need to add the dependency explicitly , it is included **in <spring-boot-starter-web>**
* To enable the XML representation , Jackson XML extension should be on classpath. To include this Add the below dependency

**<dependency>**

**<groupId>com.fasterxml.jackson.dataformat </groupId>**

**<artifactId>Jackson-dataformat.xml </artifactId>**

**</dependency>**

* To get a XML response , client needs to send appropriate ‘Accept’ header with value “application/xml” or “text/xml”
* **@RestController = @Controller + @ResponseBody**
* Spring use Http Message Converters to convert the Http Request body into domain object.

**Difference between @PathVariable and @PathParam ?**

Both are used to access the path variable comes along with {uri}. @Pathvariable is spring annotations and @PathParam is JAX-RS annotation

**Spring Security : (Spring MVC + Spring Security)**

* Spring security is an suitable framework for web applications to provide the security
* This framework provides both authentication and authorization

1. Authentication - > Identifying the User
2. Authorization -> allowing user to do some actions

\* HTTP standard itself providing some authentication mechanism , which we can use it to authenticate the user

\* HTTP Authentication :- Authenticating the user using the HTTP’s built in authentication mechanism

\* Types of Http Authentication: (it provides 2 ways to authenticate the user)

- Basic Authentication

- Digest authentication

In the above two methods , there will be a four step process

**Basic & Digest Authentication:**

1. Http Client makes a request to the web server (HTTP Request).Request method can be any. If the webserver sees that the requested resource need an authentication , it has to send the status 401(unauthorized) along with WWW-Authenticate Header.

**WWW-Authenticate :** This header assigned to realm and should have the realm directive assigned. The server will use the realm to group the different parts of the server. The client(browser) saves credentials all the realms. Whenever the browser receives the WWW-Authenticate header response with the realm already saved. It will automatically send the credentials to the server without the knowledge of the user. The browser also sends authorization request for the URL’s deeper than a level whose realm and creadentilas are known. This will create the session among the URI’s belong to the same realm.

**Format: WWW-Authenticate: realm=”videos”**

**Digest Authentication: Header Details:**

**WWW-Authenticate :** The header is assigned to realm , qop(quality of protection),

Nonce, stale, opaque , domain and algorithm directives.

**Realm:** works same as in Basic Authentication

**qop(quality of protection) :** it can have the values “auth” or “auth-int” or

“auth,auth-int”. It is an optional directive . it prevents the replay attack.

**nonce:** A unique value is generated for every 401 response. It is calculated using

the time and other values. It has expiry time and has a limit how many times it can be used. This prevents replay attack.

**Stale:** it is set to true while the client send the invalid nonce with Authorization

Request. Then the client needs to send the request with new nonce. It is an optional directive.

**Algorithm:** it is an optional directive. It can have values of any MD5 or MD5-sess

**domain:**  it is an optional directive. It can have values of list of url separated by

commas. The list indicates that all have the same credentials as the request url.Browser will save this list and credentilas. So when requesting the any of these url browser automatically send the authorization header instead of sending as normal request. The list can be cross domain.

**Opaque:** it is an optional directive. Server sends an string with this directive and

The client needs to send back unchanged. This is user to share the state information around. This can be string of base64 or hexadecimal data.

**Format:**

**WWW-Authenticate: Digest Reals=”videos” qop=”auth,auth-int” nonce=”jd839ud9832duj329u9u8ru32rr8u293ur9u329r”**

1. Once the client received the above status code and header , it display the dialog box to take the username and password as input and the client send these details with Autharization header.

**Basic Authorization :**

**Authorization :** Browser sends an username and password in this header. Username and password are joined together with colon and encoded with Base64 encoding method.

**Format:** **Authorization : Basic Do89GHt (encoded username:password)**

**Digestive Authorization:**

**Authorization:** This header should assigned to username , realm, qop,nonce,uri,nc,response and cnonce directives.

**Username:** entered by the user in dialog box

**realm:** same realm provided by the server

**nonce:** provided by server

**uri :** uri of the resource which needs to be accessed.

**qop:** Selected qop that received from server.

**nc:** This must be specified when the qop directive is sent. And this should not be sent if qop directive is not send be server along with WWW-Authenticate header.

It holds the hexadecimal count of number of request the client has send with the give nonce value. The purpose of this request is to allow the server to detect the repaying of the same request.

**cnonce:** it is a unique value generated by client. This prevents the plain text attacks. This must be sent if qop is sent by server and must not be sent qop is not send by server.

**response:** credentials are hashed and assigned to this directive. The way the credentilas are hashed is based on the algorithm and qop value.

**Format:**

**Authorization: Digest username=”vasantha” realm=”videos” nonce=” jd839ud9832duj329u9u8ru32rr8u293ur9u329ru” uri=”path/to/resource” qop=”auth” nc=”0000001” cnounce=”** **jdoiwj0d32jd23d239di9032id9i2930idik329di9i” response=”** **JHDU32D8J328UD9823UD8IDUIOSUDO”**

1. If the creadentials are correct then the server send 202 status code along with Authorization-info header.

**Basic Authentication:**

**Authentication-info :** This header is optional. Some web server send this header with information about session and future authentication requests.

Some organizations use proxy servers to authenticate users before accessing any web server resources. The same mechanisms used in the proxy servers with different status codes.

Response Status : 407 is used instead of 401.

Proxy-WWW-Authenticate : it is used instead of WWW-Authenticate.

Proxy-Authorization: it is used instead of Authorization.

Proxy-Authentication-Info : it is used instead of Authentication-Info.

**Digest Authentication:**

**Authorization-info:** This header is sent by the server if authentication success. This header is used to pass the information to the client about the next authentication request.

1. If the client sends wrong creadentials, server sends 401 status again.

Digest authentication is more secure than basic authentication because Basic authentication send credentials along with each request in form of plain text.

So it should be only used over an encrypted transport layer such as HTTPS. Digest authentication is suitable for unencrypted layer such as HTTP.

Remember-Me authentication:

It provides an way identify the user between sessions. It provides an capability to the browser to remember the user between sessions. It is accomplished by sending cookie to the browser after the successful authentication. So when the browser detects cookies for the next session, then it causing the automated login.

There are two remember-me implementations:

1. **Hashing based Token approach :** This method uses hashing based token to achieve remember-me functionality. The cookie is composed of hash values and sent it the browser.
2. **Persistent Token approach :** This method uses database to store the tokens . Database should have the persistent login table. It can be accomplished by JdbcTokenRepositoryImpl.

**CSRF Support:**

- Spring provides good CSRF Support. CSRF token are strings that is automaticalley generated and attached to the from generated from the server.

- Cross Site Request Forgery can be done by evils web site to steel information from original client request and do some fake transaction on server . The should have some extra information in the request to differentiate the evils request and original request

-

* First we need to create the security configuration , which creates the servlet filter(springSecurityFilterChain) which is responsible for all the security (validating the url , validating the submitted username and password and redirecting to the login form , etc.)
* We need to implement the **WebSecurityConfigurerAdapter** class from security config package

**<properties>**

**<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>**

**<spring.security.version>4.0.4.RELEASE</spring.security.version>**

**</properties>**

**<dependency>**

**<groupId>org.springframework.security</groupId>**

**<artifactId>spring-security-config</artifactId>**

**<version>${spring.security.version}</version>**

**</dependency>**

**Spring Multithreading:**

* Multithreading in spring can be achieved by using the threadpool.
* Thread poll can be configured in spring configuration as
* @Configuration
* public class ThreadConfig {
* @Bean
* public TaskExecutor threadPoolTaskExecutor() {
* ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
* executor.setCorePoolSize(4);
* executor.setMaxPoolSize(4);
* executor.setThreadNamePrefix("default\_task\_executor\_thread");
* executor.initialize();
* return executor;
* }
* ThreadPoolTaskExecutor class is available in org.springframework.scheduling.concurrent package

**Hibernate – Validator:**

* This framework provides some constraints on the entity fields
* Such as @NotNull , @NotEmpty and @NotBlank
* **@NotNull :-** The fields annotated with @NotNull should be not null ,but it can be empty
* **@NotEmpty :**- The fields annotated with @NotEmpty should not be null and length should be greater than zero
* **@NotBlank :**- The fields annotated with @NotBlank should not be null and trimmed length should be greater than zero

**Java Memory Management:**

Java memory areas divided into,

1. Heap :- object and instance variables are stored . String are created in String Pool of Heap space
2. Stack :- local variables and method calls are stored in stack each thread has its own copy of local variables. Variable references either primitive reference or object references are also stored .
3. Method Area(Permanent Generation Space(permGen)) :- static variables and static and non-static methods stored here. All class level level data to be here.

From Java 1.8 static variables are stored in heap itself.

**How to deal with out of memory error:**

* It happened due to the lack of memory for the object
* Java.lang.Object -> java.lang.Throwable -> java.lang.Error ->java.lang.VirtualMachineError

->OutOfMemoryError

- To overcome the OutOfMemoryError , we need to examin the java memory management scheme

**What is hibernate dirty checking ?**

Dirty checking is feature in hibernate to monitor the state of the object if any associated with session. Hibernate supports automatic dirty checking. The object must be in cache for the dirty checking takes place. Hibernate monitors the objects, if any changes happened it automatically fire the sql query even though we are not explicitly calling. The transaction should be commited for the dirty checking to be work.

It will synchronize the object from the persistent context to database. The monitoring the object and committing the changes happened at the time of flush. This happens in hibernate with the use of Persistent Context. Hibernate loads all entities from database copy of these entity will be stored in Persistentcontext. Hibernate checks the objects into the entity on persistent context to see if any changes happened and fire the update on modified objects. At end of the transation hibernate aquires the proper locks on tables and do the updation and release the locks.

To customize the dirty checking , we can use the hibernate intercepters , findDirty() method.

Whats is thread dump ?

What is Thread factory?

How to ensure the code quality?

What is code coverage?